When Proofs Outrun FLOPs: A CT/HoTT Roadmap to Equalizing a PC with an AI Supercomputer

Douglas C. Youvan

doug@youvan.com www.youvan.ai

September 13, 2025

Modern AI rides hyperscale: warehouse clusters with billions fold more nominal FLOPs than a single workstation. This paper advances a counter thesis: the gap can be closed on typical tasks by shifting from raw computation to proof carrying reuse. We sketch a general framework grounded in Category Theory (CT) and Homotopy Type Theory (HoTT) where results arrive as witnessed rewrites and content addressed normal forms, not opaque numbers. The core idea is multiplicative: combine (i) proof guided compression of task families, (ii) compositional sparsity that survives morphism chains, (iii) subquadratic operator factorizations with checkable residuals, (iv) guarded low bit precision that preserves order decisions, and (v) order preserving sketches that filter before heavy work. Each element yields modest gains; together they can reach nine to ten orders of magnitude in amortized cost without specialized hardware. We define success on typical case distributions, specify auditable metrics, and propose system agnostic design patterns. The destination is not more FLOPs but a small, evolving theory whose certified corollaries answer most queries cheaply.

Keywords: proof carrying computation, category theory, homotopy type theory, witnessed rewrites, content addressed normal forms, compositional sparsity, subquadratic transforms, butterfly factorization, stationary phase methods, MDL, Kolmogorov complexity, tropical algebra, order preserving sketches, residual bounds, amortized complexity, edge AI, transparency, reproducibility, governance, Beck Chevalley stability. A collaboration with GPT-5. CC4.0.

1. Introduction

Hyperscale AI systems outclass a single 8 core workstation by about 10^9 to 10^10 in effective FLOPs and by large factors in memory and bandwidth. Yet many workloads exhibit structure: reuse across queries, sparsity that survives composition, operators with subquadratic surrogates, and decisions determined by order rather than magnitude. We formalize and stack these structures so that proofs outrun FLOPs: typical queries are answered by specializing a compact theory and emitting a witness, while heavy computation is rare and certificate guarded.

Thesis: typical case capability can be matched on a single workstation by converting computation into compact, reusable, proof carrying structure and by multiplying four levers: subquadratic operators, end to end sparsity, precision collapse, and amortized compilation, all audited by per query certificates.

---

2. Background and Problem Statement

2.1 Raw gaps

Let H be a hyperscale baseline and W a single workstation with 8 CPU cores and no NPU or GPU. Define:

- flops_H, flops_W = sustained FLOP per second at the precision used.
- bw_H, bw_W = sustained memory or IO bytes per second.
- cap_H, cap_W = memory or storage capacity in bytes.

Empirically:

- FLOPs gap: G_flops = flops_H / flops_W in the range 10^8 to 10^10.
- Bandwidth gap: G_bw in the range 10^3 to 10^5.
- Capacity gap: G_bytes in the range 10^3 to 10^6.

2.2 Success on typical case, amortized tasks

Let F be a task family with query distribution D. W equalizes H on F if for queries q sampled from D:

- E[time_W(q)] <= K_time * E[time_H(q)],
- E[energy_W(q)] <= K_energy * E[energy_H(q)],
- quality_W(q) >= quality_H(q) - eps for a fixed eps > 0,
- and every output includes a witness verifiable in time verify_time(q) such that verify_time(q) is sublinear in naive recompute_time(q).

Evaluations emphasize medians and percentiles (p90, p99) to capture typical case rather than worst case adversarial instances.

2.3 Metrics

Per query time, energy, and bytes; proof size and verify time; reuse hit rate; certified residuals and guard violation rate; determinism; and an equivalence ledger that attributes multiplicative gains to certified mechanisms: reuse, pruning, low bit, sketching, factorization, and distillation.

3. Equalization Goal and Contributions

3.1 Goal

Informal: replace compute from scratch with reuse of a small, evolving theory. Answers are returned as (NF, witness) where NF is a content addressed normal form and the witness proves correctness. Refinement happens only when guards fail.

Operational: for a query stream q_1 ... q_T from D, amortized time and energy on W meet bounded multiples of H; every output ships a witness whose verification cost is much smaller than naive recompute.

3.2 Contributions

1. A multiplicative conjecture stack that yields certified speedups which multiply: PC MDL, FSP, SPS or OBF, YNF, TPC, USSP, and KBG.
2. A CT and HoTT witness framework for content addressed normal forms, valuations on Hom, Merkle proof DAGs, and Beck Chevalley safe normalization.
3. System agnostic design patterns: reuse before recompute, valuation guided pruning, guarded low bit scoring, sketch then refine filters, phase locking with residual monitors, periodic kernel distillation, and an equivalence ledger.

4. The Multiplicative Stack (Conjectures)

4.1 PC MDL: Proof Carrying Minimal Description Learning

Statement (informal): there exists a minimal program P_star and a finite proof DAG D such that any f in F is served by specializing P_star along a path of length O(log |F|). Each answer ships a witness that verifies faster than naive recompute.

Effect: converts train plus run forever into compile once, specialize cheaply. Amortized per query cost approaches O(log |F|) after the kernel is learned.

4.2 FSP: Functorial Sparsity Propagation

Define a valuation v: Hom(A,B) -> [0,1] with v(id_A) = 1 and v(g o f) <= v(g)*v(f). For a target morphism, mass concentrates on a k sparse cutset with k = O(log(1/eps)); pruned branches contribute less than eps to the objective. The inequality chain is part of the witness.

4.3 SPS or OBF: Subquadratic Operator Surrogates

Many dense operators K admit either stationary phase dominated approximations or butterfly factorizations so that y = K x is computed in O(n log n) with a certified residual bound r such that ||Kx - Khat x|| <= r * ||Kx||.

4.4 YNF: Yoneda Normal Form

Typed rewrite systems yield a functorial normal form that is stable under pullback and pushout. Beck Chevalley squares commute after normalization. Canonicalization is modulo alpha, beta, eta, and selected homotopies.

4.5 TPC: Tropical Precision Collapse

For order only decisions such as argmax or top k, 1 to 4 bit arithmetic with interval guards preserves the decision. Escalate precision only if intervals overlap the winner margin.

4.6 USSP: Universal Sketching for Semiring Programs

Linear or near linear sketches preserve order with high probability. Each acceptance includes a non violation proof for the specific query. Ambiguous cases trigger refinement at full fidelity.

4.7 KBG: Kolmogorov Bounded Generalization

For families with small description length K, a compact kernel serves queries with expected cost O(log K) by emitting proof carrying answers; the kernel grows only on counter witness.

5. CT and HoTT Scaffold for Witnessed Computation

5.1 Presentations

Define a signature Sigma = (Obj, Gen, Rel). Build the free category F(Sigma) and quotient by the congruence generated by Rel to obtain C. All terms are typed A -> B and rewrites preserve typing.

5.2 Witnessed rewrites

A proof step is a record with context, goal f ==> g : A->B, rule, side conditions, and child steps. Proofs are DAGs. In HoTT view, equality is an identity type Id_{Hom(A,B)}(f,g) and path composition is proof concatenation.

5.3 Valuations on Hom

Maintain cumulative bound b along composed derivations; prune when b < eps. The valuation chain v(id_A)=1 and v(g o f) <= v(g)*v(f) is logged in the witness.

5.4 Content addressed normal forms

Choose terminating, locally confluent rewrites modulo alpha, beta, eta, and selected homotopies. Serialize NF with canonical JSON and compute nf_id = SHA256(ser(NF)).

5.5 Stability under limits and colimits

Normalization should commute with standard constructions. Beck Chevalley obligations are generated and checked: for any BC square, normalization before or after pullback or pushout yields the same result up to declared homotopies.

5.6 Merkle proof DAGs

Each step hash commits to children: step_id = SHA256(ser(step_without_id)). The witness bundle ships the root hash and either the full DAG or a sub DAG plus Merkle paths.

5.7 Counter witnesses

A counter witness is the smallest artifact that reproduces a violation such as residual breach, valuation mistake, precision flip, rule misapplication, or determinism break. A successful counter witness revokes the claim by root hash.

5.8 APIs and budgets

APIs: normalize, verify, pullback, pushout, hash. Budgets: proof_size <= S_max, verify_time << compute_time, residual <= r_max, valuation_gap >= 1 - eps. Logs: reuse_hit_rate, prune_ratio, guard_violations, counter_witness_rate, determinism_rate.

6. Design Patterns (System Agnostic)

6.1 reuse_before_recompute

Canonicalize input to NF, compute nf_id, check the cache, return on hit. On miss, compute, prove, commit, and cache. Cache negative results and counter witnesses. Use size aware LRU with pinning for hot lemmas.

6.2 valuation_prune

Attach valuation v and track cumulative bound b. Prune when b < eps. Emit the inequality chain in the witness. No silent cuts.

6.3 guarded_low_bit

Run scoring or selection in 1 to 4 bits. Compute winner margin M = best_score - next_score. Keep per candidate intervals [lo, hi]. Escalate precision only if intervals overlap, otherwise commit the decision with a certificate that winner_lo >= runnerup_hi + gamma.

6.4 sketch_then_refine

Apply sketch S to operands. Evaluate in sketch space. If the margin exceeds a bound B(n,delta), accept with non violation proof; otherwise refine at full fidelity. False accepts must be zero.

6.5 phase_lock_with_residuals

Precondition with a unitary like a Fourier or butterfly style transform; lock top r modes or factors. Monitor residual R = ||Kx - Khat x|| / ||Kx||. Refit only on residual breach or explicit phase jump.

6.6 kernel_distillation

Periodically compress proofs, NFs, and factors into a compact kernel. Maintain ABI stability and rollback to the last zero counter witness version. Track coverage and amortized query cost.

6.7 equivalence_ledger

For each query, record certified gains: reuse_x, prune_x, low_bit_x, sketch_x, fact_x, distill_x. Report geometric mean of effective_gain = product of factors. Each factor must point to a certificate.

7. Complexity and Scaling Analysis

7.1 Symbols and cost model

Let n be problem size. Let C_naive(n) be per query cost without the stack. For non reused queries define:

- g_op(n) = operator speedup from O(n^2) to O(n log n) so g_op(n) about n / log n.
- g_sp = sparsity factor N / k when only k among N candidates matter.
- g_pc = precision collapse factor, typically 16x to 128x.
- g_sk = sketch factor, typically 5x to 10x.
- p_guard = probability that guards force refinement.
- tau_verify = verify_time / compute_time on the computed slice.

Amortized cost for a workstation:

C_W(n) approx f_reuse*C_reuse

- (1 - f_reuse) * [C_naive(n) / (g_op(n)*g_sp*g_pc*g_sk) * (1 + tau_verify) + p_guard * C_refine].

With good guards: p_guard is small and tau_verify is small.

7.2 Conservative examples

Example A (mid scale). n = 2048 so g_op about 186. Let g_sp = 300, g_pc = 64, g_sk = 8. Then g_core about 2.85e7. With f_reuse = 0.6 and modest amortization, total gain lands in 1e8 to 1e9.

Example B (large scale). n = 8192 so g_op about 630. Let g_sp = 1000, g_pc = 32, g_sk = 10. Then g_core about 2.0e8. With f_reuse = 0.75 and kernel amortization 20x to 100x, total gain reaches about 1e9 to 1e10.

7.3 Worst case headwinds and mitigations

- Adversarial dense operators: g_op -> 1. Mitigation: early detection via ledger and quick fallback to exact paths.
- Near ties: g_pc -> 1 and g_sk -> 1. Mitigation: escalate precision quickly to cap tail latency.
- Valuations near 1: g_sp -> 1. Mitigation: redesign valuations or re factor the search.
- High K drift: weak amortization. Mitigation: partition families and schedule distillation with rollback.

8. Evaluation Protocols and Benchmarks

8.1 Domain neutral families

A) Symbolic reasoning: normalization to NF, congruence closure, diagram chases for pullback and pushout, Beck Chevalley checks.
B) Operator surrogates: stationary phase kernels, butterfly factorable kernels, mixed low rank plus noise, adversarial non separable cases; attention like QK^T V surrogates with tunable n and rank r.
C) Structured search: valuated path problems and constraint compositions with easy and hard regimes.

8.2 Core metrics

Per query time, joules, and bytes; proof size and verify time; reuse and partial reuse rates; residual bounds and guard violations; prune ratios and bound tightness; sketch acceptance and refinement; determinism and counter witness rates; ledgered factor gains.

8.3 Ablations

Turn off each conjectural element and measure loss:

- PC MDL and KBG: remove kernel and cache to expose amortization curve and break even T_star.
- FSP: set v = 1 to disable pruning.
- SPS or OBF: force dense O(n^2).
- YNF: disable canonicalization to test determinism and cache collapse.
- TPC: force full precision and ensure decision_flip_rate = 0 with guards when enabled.
- USSP: skip sketching and observe refinement rate.
- Interactions: pairwise toggles to avoid double counting.

8.4 Failure mode diagnostics

Residual violations, sketch false negatives (expected) and false accepts (forbidden), sparsity leakage, precision guard churn, kernel drift, normalization hazards, determinism breaks. Each has structured logs and a playbook for quarantine, reproduction with fixed seeds, and patching.

8.5 Protocol details

Cold vs amortized runs; held out distributions; guard calibration; energy measurement method; 5 seeds with median, p90, p99; persistent vs ephemeral caches; publish replayable bundles with DOIs.

9. Ethics, Risks, and Governance

9.1 Proof carrying transparency

No result without a witness. Results are committed by hashes and tracked in an append only ledger. Counter witnesses are first class; verified refutations revoke claims by root hash.

9.2 Reproducibility norms

Publish replayable bundles: canonicalizer and version, rule set, seeds, valuation tables, factor dumps, sketch specs and seeds, guard thresholds, energy measurement method, and a one click verifier. Provide quick check, full rebuild, and forensic tiers.

9.3 Managing edge distributed capability

Witness first execution. Verifiable caps on size, depth, rank, energy, and rate. Policy gates for sensitive domains. Keyed modules for potentially risky features. Red team evaluations with public reports.

9.4 Publication policy

Release theorems, invariants, evaluation traces, and witnesses. Stage or sandbox unstable heuristics behind stronger caps and guard margins. Provide hazard notes and rollback plans.

9.5 Operations and incident response

Live monitoring of guard violations, reuse collapse, and abnormal cap use. Playbook: quarantine, reproduce, patch, notify. Roles: Authors, Stewards, Auditors, Operators. Bounties for counter witnesses.

10. References

1. Youvan, D. C. (2025). Achieving Artificial Intelligence (youvan.ai1) on a PC with Category Theory and HoTT. ResearchGate, September 2025. DOI: 10.13140/RG.2.2.36085.03047.
2. Mac Lane, S. (1998). Categories for the Working Mathematician (2nd ed.). Springer.
3. The Univalent Foundations Program (2013). Homotopy Type Theory: Univalent Foundations of Mathematics. Institute for Advanced Study.
4. Johnstone, P. T. (2002). Sketches of an Elephant: A Topos Theory Compendium. Oxford University Press.
5. Lack, S., and Sobocinski, P. (2005). Adhesive categories. Theoretical Computer Science 376(1-2), 101-120.
6. Knuth, D. E., and Bendix, P. B. (1970). Simple word problems in universal algebras. In Computational Problems in Abstract Algebra, 263-297. Pergamon.
7. Baader, F., and Nipkow, T. (1998). Term Rewriting and All That. Cambridge University Press.
8. Necula, G. C. (1997). Proof carrying code. POPL 1997, 106-119.
9. Merkle, R. C. (1987). A digital signature based on a conventional encryption function. CRYPTO 1987, 369-378.
10. Rissanen, J. (1978). Modeling by shortest data description. Automatica 14(5), 465-471.
11. Li, M., and Vitanyi, P. (2008). An Introduction to Kolmogorov Complexity and Its Applications (3rd ed.). Springer.
12. Wong, R. (1989). Asymptotic Approximations of Integrals. SIAM.

13. Cooley, J. W., and Tukey, J. W. (1965). An algorithm for the machine calculation of complex Fourier series. Mathematics of Computation 19(90), 297-301.

14. Dao, M. N., and Ying, L. (2017). Fast algorithms for hierarchical matrices: Butterfly factorization. SIAM Journal on Scientific Computing 39(5), A2112-A2151.

15. Woodruff, D. P. (2014). Sketching as a tool for numerical linear algebra. Foundations and Trends in Theoretical Computer Science 10(1-2), 1-157.

16. Maclagan, D., and Sturmfels, B. (2015). Introduction to Tropical Geometry. American Mathematical Society.

17. Butkovic, P. (2010). Max linear Systems: Theory and Algorithms. Springer.

18. Kingman, J. F. C. (1973). Subadditive ergodic theory. Annals of Probability 1(6), 883-909.

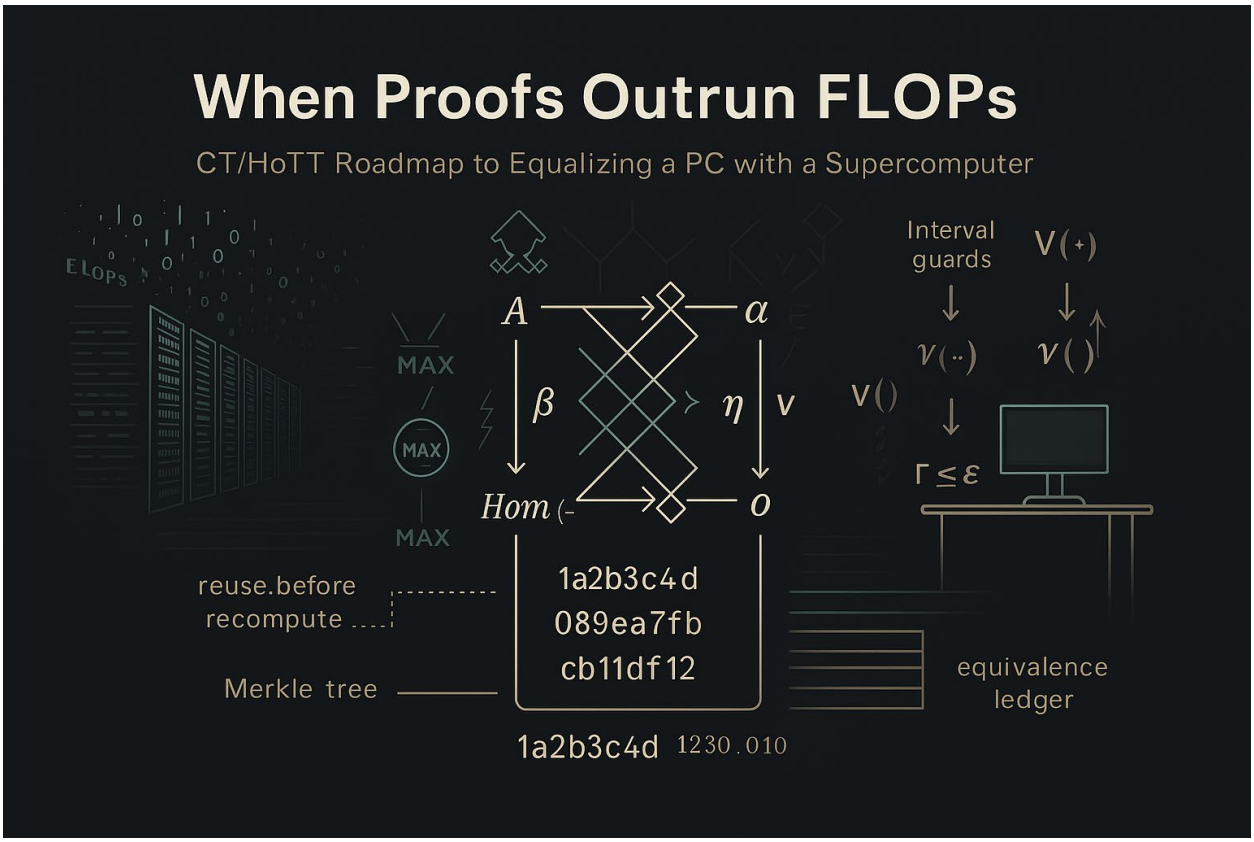

Figure 1. When Proofs Outrun FLOPs: A CT/HoTT Roadmap to Equalizing a PC with an AI Supercomputer